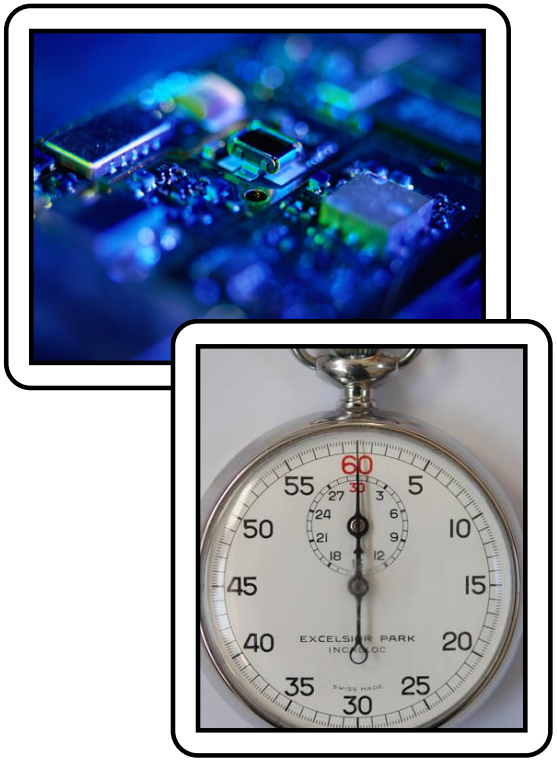
Complete, High-Assurance Determination of Loop Bounds and Infeasible Paths for WCET Analysis

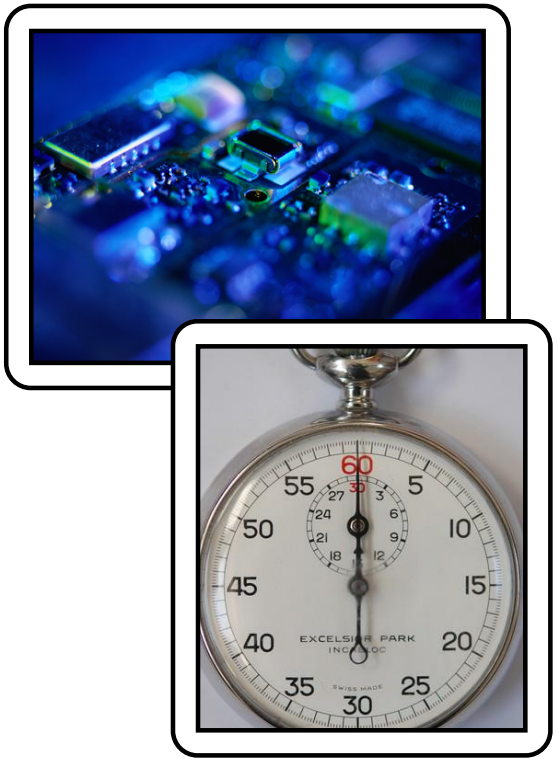
Thomas Sewell, Felix Kam, Gernot Heiser

10 May 2016

WCET Analysis

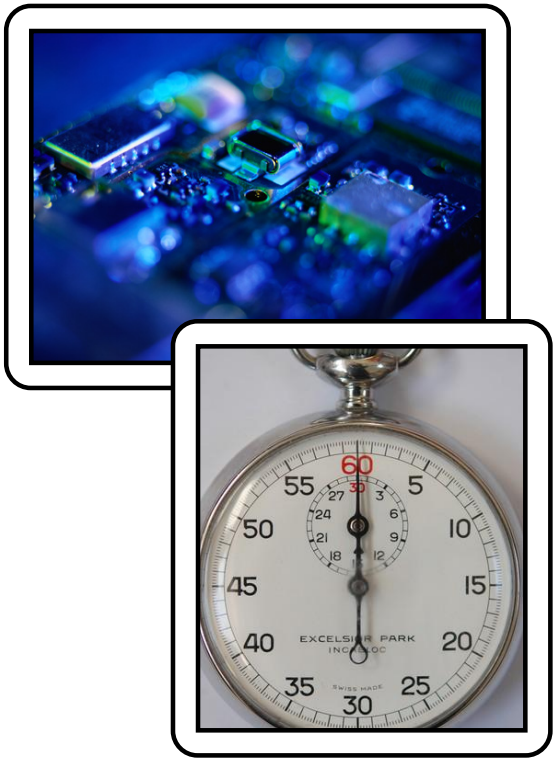


WCET Analysis




```
0170:   e1a0e00d   mov    lr, sp
0174:   e28dc008   add    ip, sp, #8
0178:   e88e000c   stm    lr, {r2, r3}
017c:   e1a03002   mov    r3, r2
0180:   e202200e   and    r2, r2, #14
0184:   e352000e   cmp    r2, #14
0188:   e88c0003   stm    ip, {r0, r1}
018c:   1203100f   andne  r1, r3, #15
```

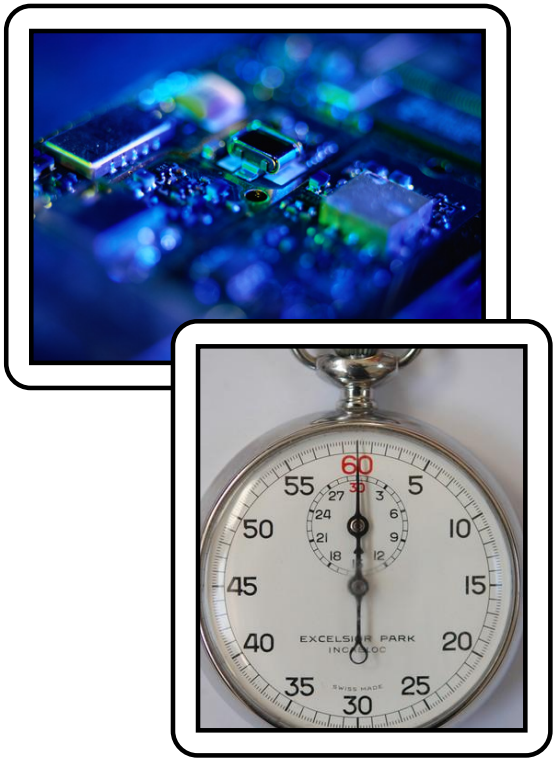
WCET Analysis



```
0170:   e1a0e00d   mov    lr, sp
0174:   e28dc008   add    ip, sp, #8
0178:   e88e000c   stm    lr, {r2, r3}
017c:   e1a03002   mov    r3, r2
0180:   e202200e   and    r2, r2, #14
0184:   e352000e   cmp    r2, #14
0188:   e88c0003   stm    ip, {r0, r1}
018c:   1203100f   andne  r1, r3, #15
```

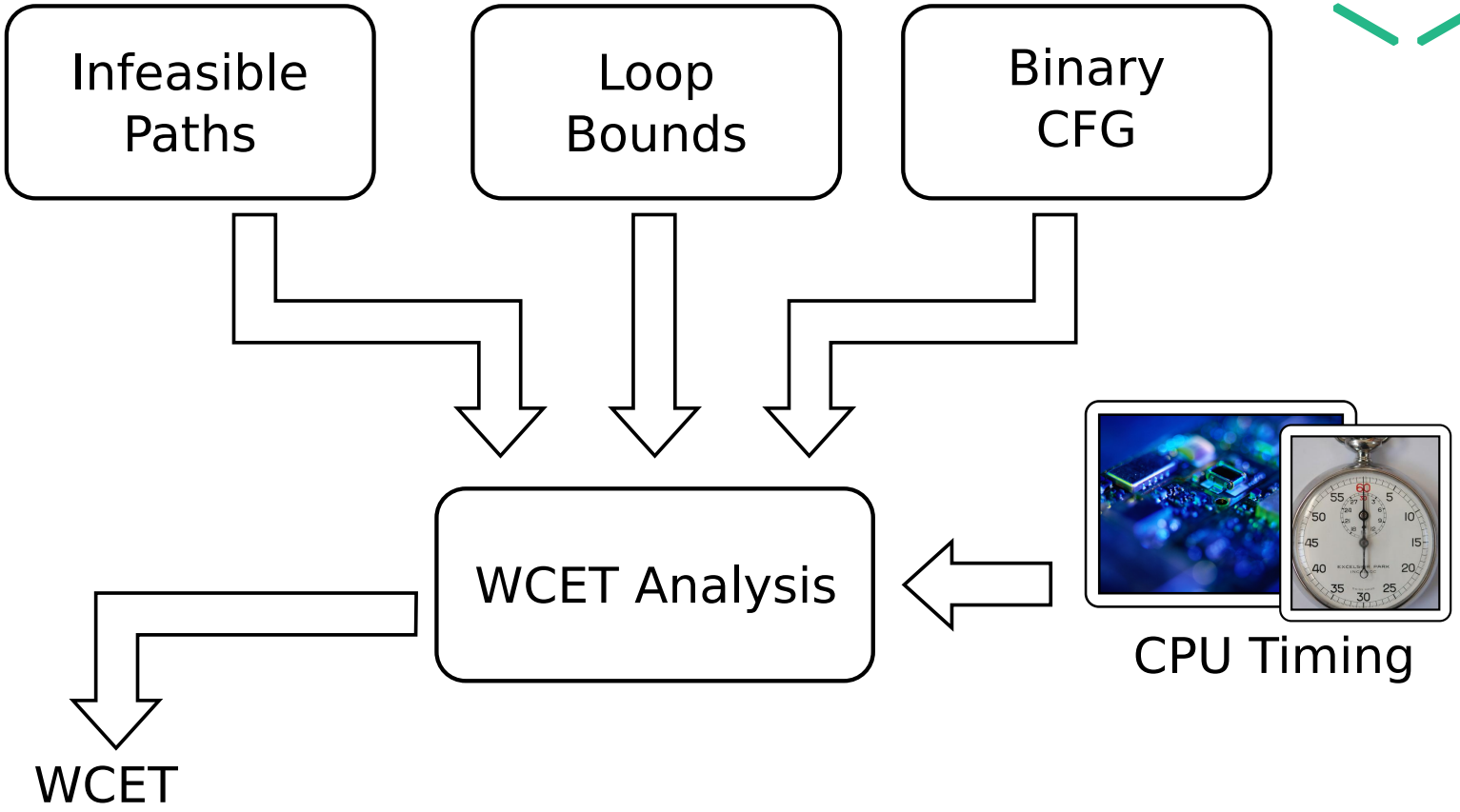
- Instruction Timing

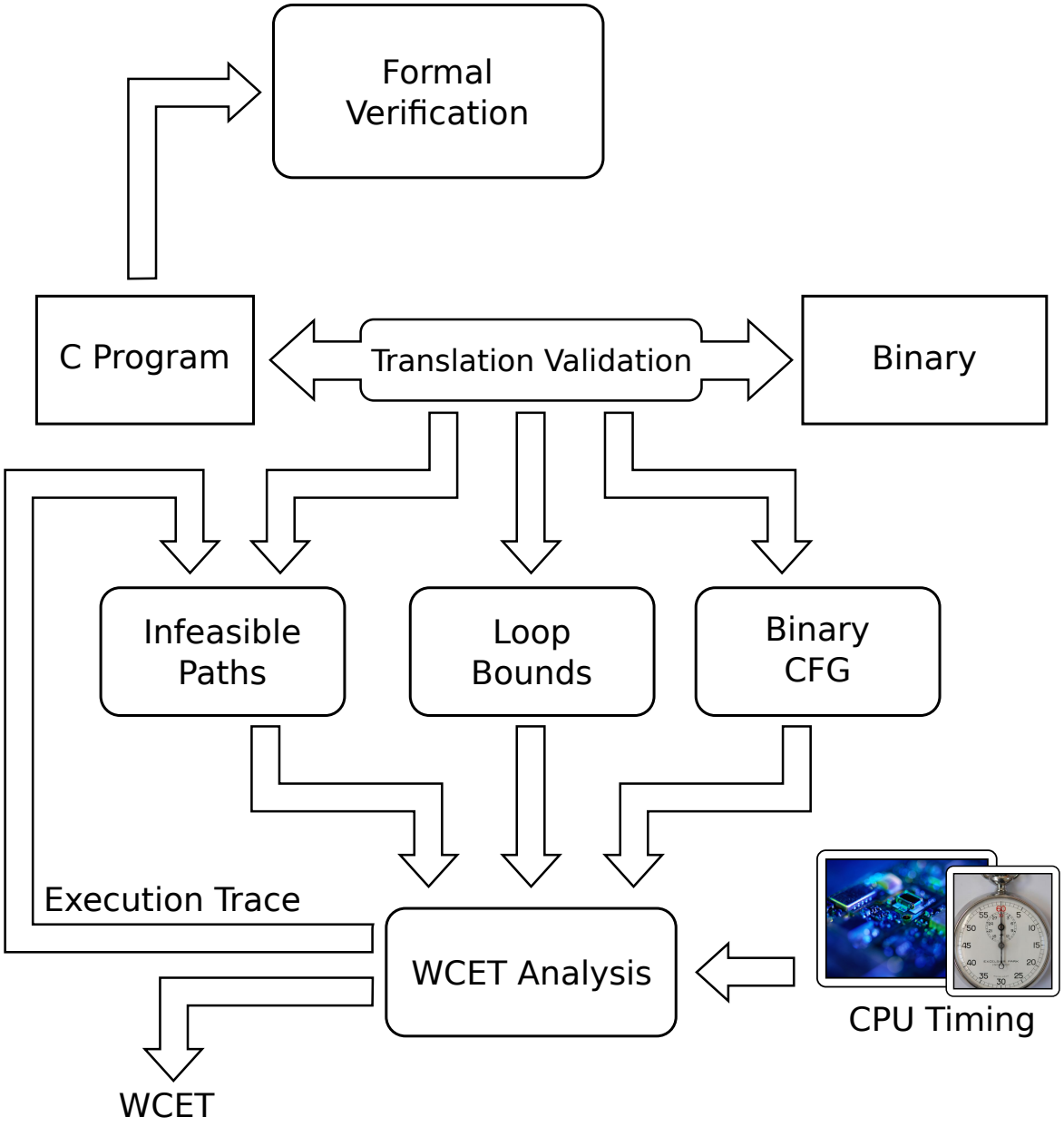
WCET Analysis

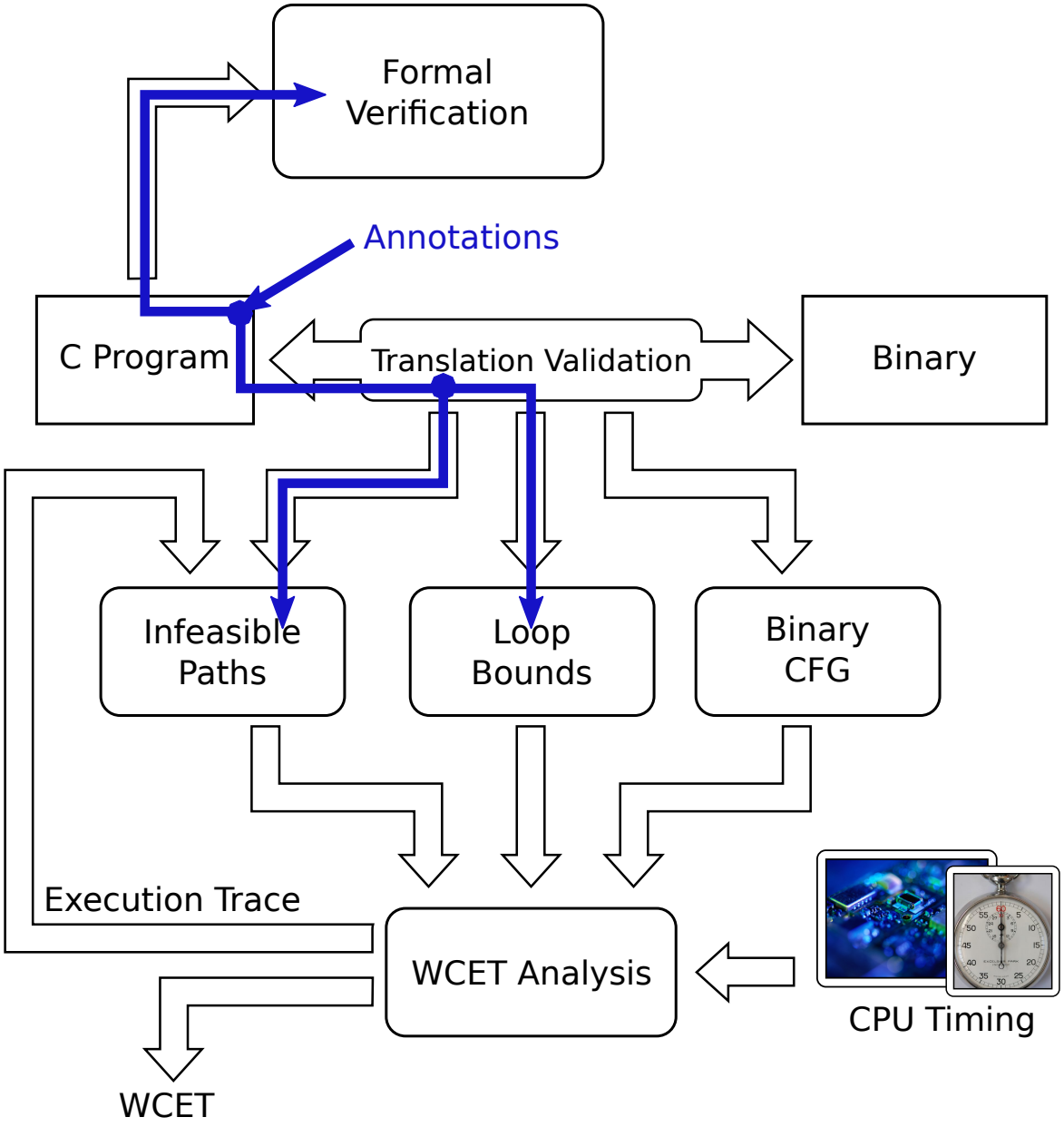


```
0170:   e1a0e00d   mov    lr, sp
0174:   e28dc008   add    ip, sp, #8
0178:   e88e000c   stm    lr, {r2, r3}
017c:   e1a03002   mov    r3, r2
0180:   e202200e   and    r2, r2, #14
0184:   e352000e   cmp    r2, #14
0188:   e88c0003   stm    ip, {r0, r1}
018c:   1203100f   andne  r1, r3, #15
```

- Program Analysis
- Instruction Timing


Infeasible Paths
Loop Bounds
Binary CFG
WCET Analysis
CPU Timing
WCET


Formal Verification
C Program
Translation Validation
Binary
Infeasible Paths
Loop Bounds
Binary CFG
Execution Trace
WCET Analysis
CPU Timing
WCET


Formal Verification
Annotations
C Program
Translation Validation
Binary
Infeasible Paths
Loop Bounds
Binary CFG
Execution Trace
WCET Analysis
CPU Timing
WCET

Timing a Verified Microkernel

Timing seL4: a formally verified microkernel.

- Proven correct to the highest level of assurance.

Security. Performance. Proof.

Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski and Gernot Heiser
"Comprehensive formal verification of an OS microkernel"
ACM Transactions on Computer Systems, Volume 32, Number 1, pp. 2:1-2:70, February, 2014

WCET analysis should be as **trustworthy** as verification.

seL4

- General-purpose, memory-protected OS microkernel.
- $\approx$ 9000 lines of C code.
- Active development.
- Features for real-time & mixed-criticality.
- Event-reactive kernel.
  - Single stack.
  - Not reentrant.
  - WCET critical to system response time.

seL4

- General-purpose, memory-protected OS microkernel.
- $\approx$ 9000 lines of C code.
- Active development.
- Features for real-time & mixed-criticality.
- Event-reactive kernel.
  - Single stack.
  - Not reentrant.
  - WCET critical to system response time.

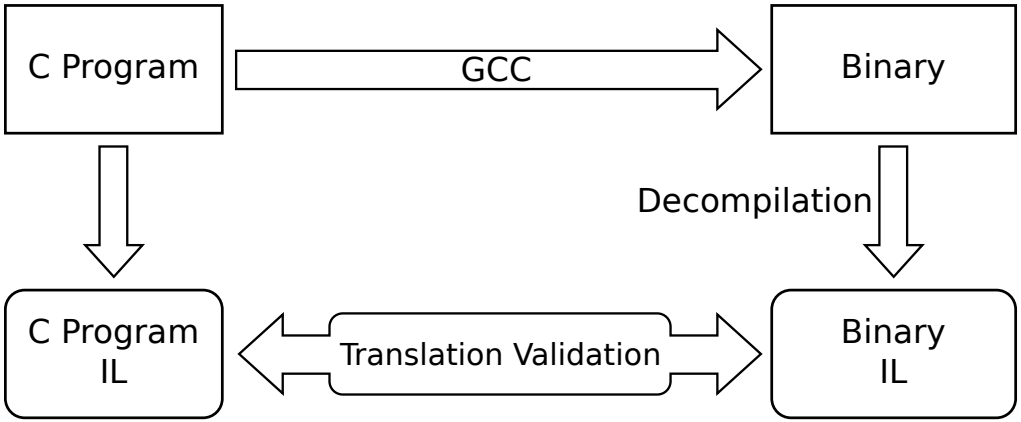
Source and proofs are open source:

http://seL4.systems/
http://github.com/seL4/seL4
http://github.com/seL4/l4v

Translation Validation

Thomas Sewell, Magnus Myreen and Gerwin Klein
"Translation validation for a verified OS kernel"
ACM SIGPLAN PLDI, pp. 471-481, Seattle, Washington, USA, June, 2013

Translation Validation Steps



Translation validation (TV) steps:

- Convert source & binary to a shared language.
  - Explicit CFG.
  - ARM semantics thanks to Cambridge, Magnus Myreen.
- Prove equivalence.
  - Powerful automatic reasoning engine & loop analysis.

Infeasible Paths

Code reachability:

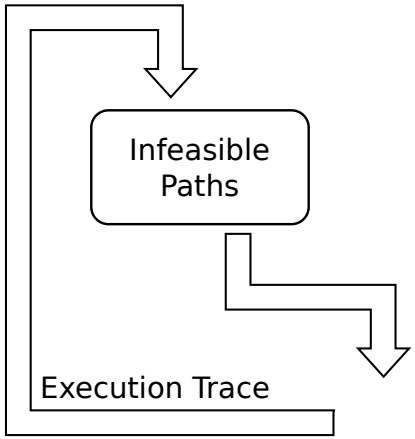
- Translation validation (TV) primitive.
- e.g. for dead code analysis.

We extend to infeasible paths.

We allow paths across function calls.

- Requires slight TV extension.

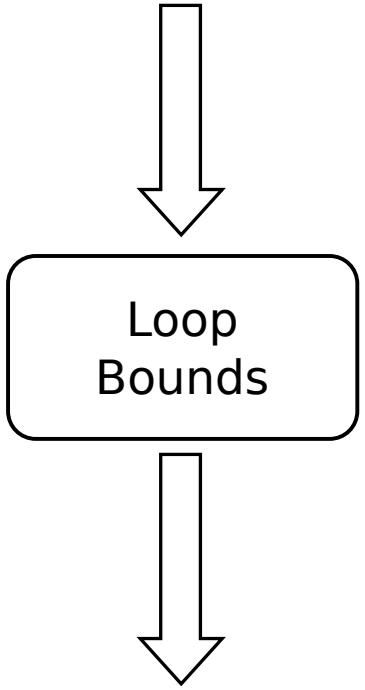
We focus on candidate WCET traces.



Loop Bounds

Loop bounds:

- Explicit strategy.
- Inductive strategy.
- TV transfer strategy.



Loop Bounds I

Explicit loop bound strategy:

- Unwind initial iterations.
- Test path feasibility.

```
for (i = 0; str[i]; i ++) {
  str2[i] = str[i];
}
---------------------------
i = 0;
if (! str[i])
  goto done;
str2[i] = str[i];
i ++;

if (! str[i])
  goto done;
str2[i] = str[i];
i ++;

...
```

Loop Bounds II

Loops with higher bounds:

- Discover linear sequences.
- Prove by induction.
- Derive loop bound.

```
for (i = 0; i < 256; i ++) {

  pt[i] = 0;
}
```

```
c00:   e3a03000   mov   r3, #0
c04:   e1a02003   mov   r2, r3

c08:   e7802003   str   r2, [r0, r3]
c0c:   e2833004   add   r3, r3, #4
c10:   e3530b01   cmp   r3, #1024
c14:   1afffffb   bne   c08
```

Loop Bounds II

Loops with higher bounds:

- Discover linear sequences.
- Prove by induction.
- Derive loop bound.

```
for (i = 0; i < 256; i ++) {
   i = i0 + δ & i <= 256
  pt[i] = 0;
}
```

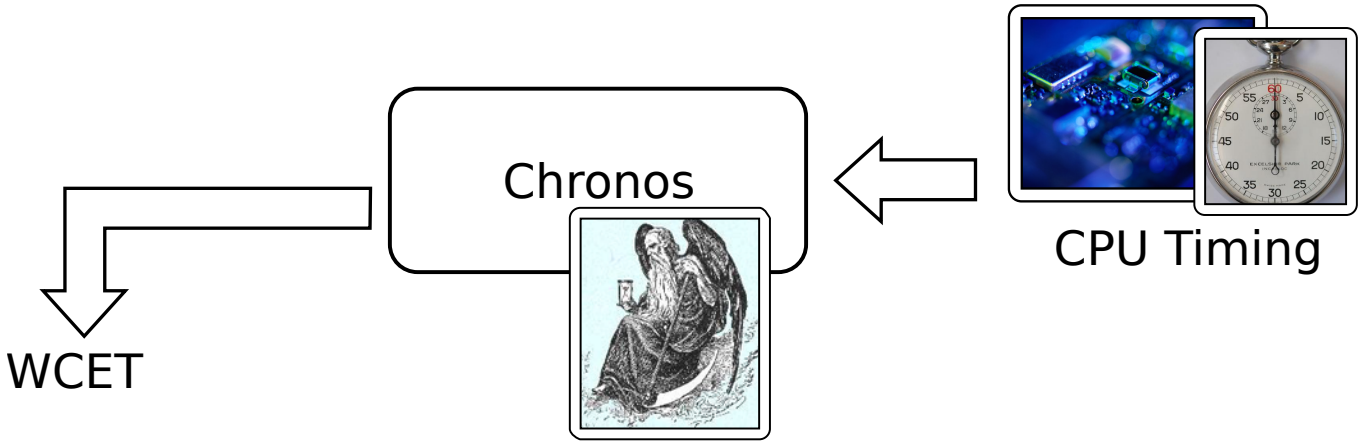
```
c00:   e3a03000   mov   r3, #0
c04:   e1a02003   mov   r2, r3
   r3 = r30 + 4δ & r3 <= 1024
c08:   e7802003   str   r2, [r0, r3]
c0c:   e2833004   add   r3, r3, #4
c10:   e3530b01   cmp   r3, #1024
c14:   1afffffb   bne   c08
```

Loop Bounds III

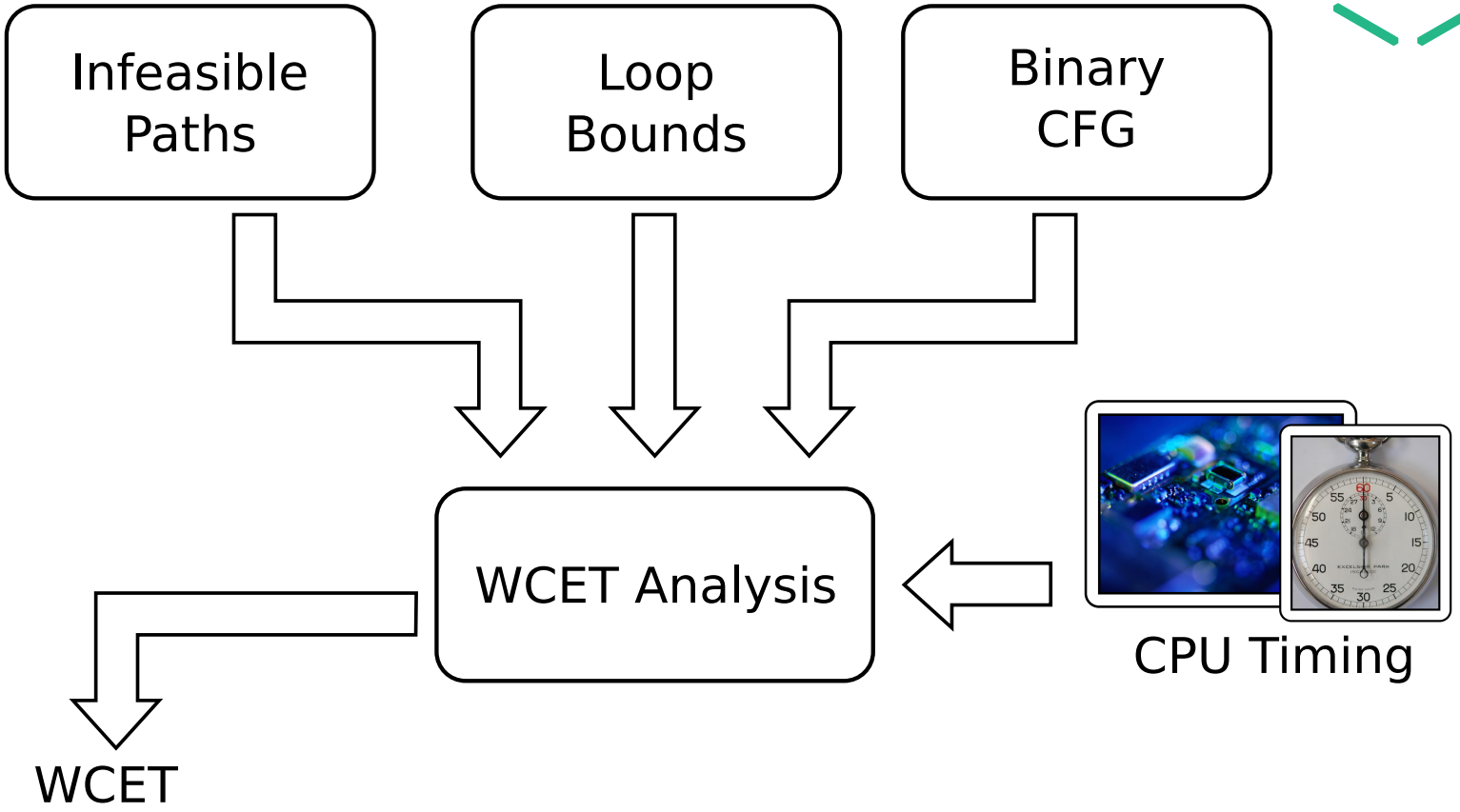
Loop bounds:

- Explicit & Inductive strategy.
  - Use C-level information implicitly.
- Translation validation (TV) transfer strategy:
  - Explicit & inductive search on C.
  - Discover & prove TV simulation relation.
  - Derive binary bound.

Chronos



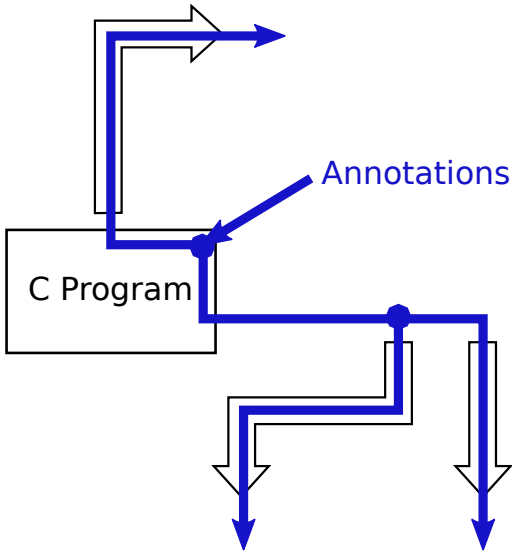
We use a modified version of NUS Chronos.
It models the Freescale IMX31, an ARM1136.


Infeasible Paths
Loop Bounds
Binary CFG
WCET Analysis
CPU Timing
WCET

Results

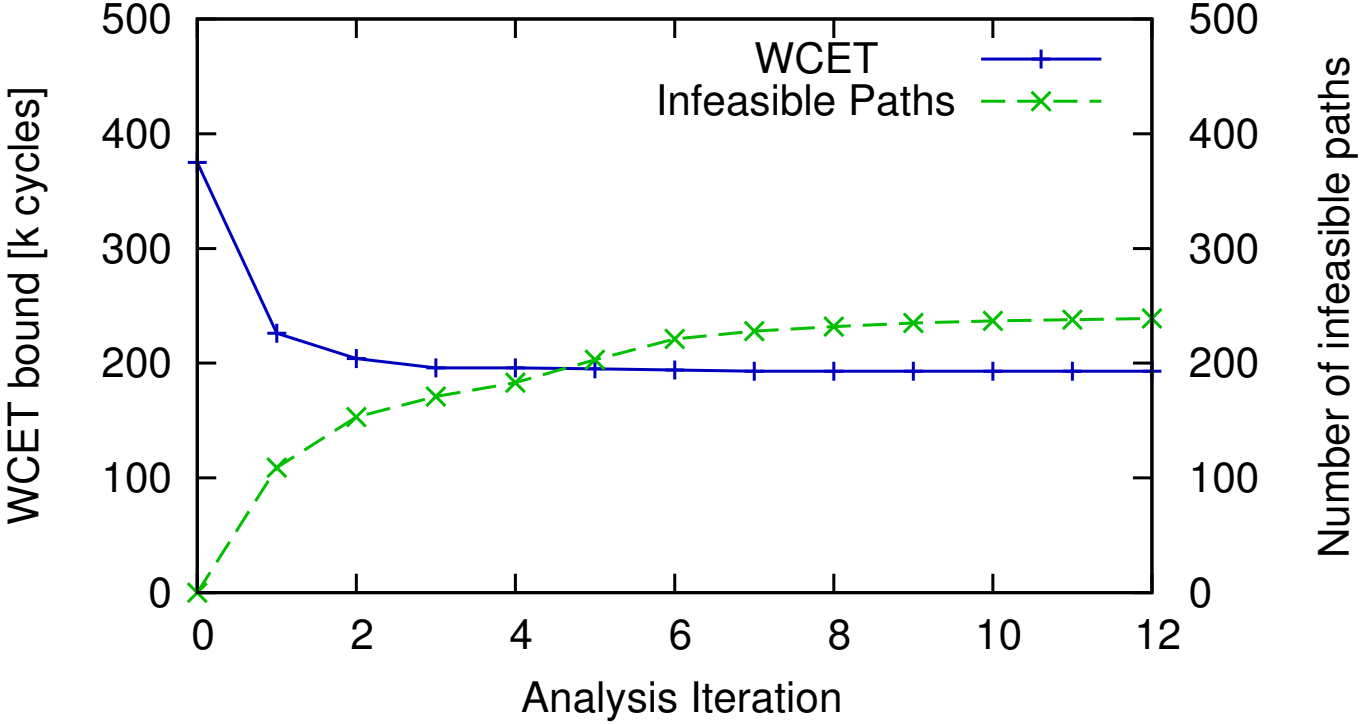
We compute the bounds of all 67 bounded loops in seL4.
(A further 5 loops have preemption points.)

We add 22 annotations to the source code.



Results II

With infeasible paths added, we measure WCET.



Annotations

Four of the 22 annotations limit a "length" field.

These annotations are **assertions** in the source.

- **Assumptions** of the translation validation (TV).
- **Obligations** of the verification.

```
exception_t
performPageInvocationRemapPTE
    (asid_t asid, ...)
{
    ...
    j = pte_entries.length;
    /** GHOSTUPD: "(\<acute>j <= 16, id)" */

    for (i = 0; i < pte_entries.length; i++) {
        pte_entries.base[i] = pte;
    }
    cleanCacheRange_PoU
        ((word_t)pte_entries.base, ...);
    ...
}
```

These assertions are now **verified**.
In 2.1.0 seL4: http://github.com/seL4/seL4

Annotation Kinds

We add four kinds of annotations:

- Limit a "length" field to 16. (4 annotations.)
  - Supports inductive strategy.
- Ensure at least 1 bit is resolved in a lookup loop. (1 annotation.)
  - Supports TV transfer strategy.
- Limit object size to 64Kib. (10 annotations.)
  - Supports inductive strategy.
  - Needed for mainline seL4.
- Prune a cleanup function at some call sites. (7 annotations.)
  - Supports infeasible paths.

Results III

We find 47 of 67 bounds (70%) with source information hidden.

We also test on the Mälardalen test suite.

Benchmark	Bounds
BS	1 / 1
BSORT100	1 / 2
COVER	3 / 3
FDCT	2 / 2
FIBCALL	1 / 1
JFDCTINT	2 / 2
STATEMATE	0 / 1

Results III

We find 47 of 67 bounds (70%) with source information hidden.

We also test on the Mälardalen test suite.

Benchmark	Bounds
BS	1 / 1
BSORT100	1 / 2
COVER	3 / 3
FDCT	2 / 2
FIBCALL	1 / 1
JFDCTINT	2 / 2
STATEMATE	0 / 1

```
int
main () {
  test ();
}
```

Comparisons

Compared to our previous work on seL4:

- Use mainline, verified seL4; WCET higher.
- Support for higher bounds.
- Function-by-function analysis.
- Source-level verified annotations.

Comparisons

Compared to our previous work on seL4:

- Use mainline, verified seL4; WCET higher.
- Support for higher bounds.
- Function-by-function analysis.
- Source-level verified annotations.

Compared to other WCET analysis:

- Similar to custom compiler approaches.
- Far more trustworthy at compiler and annotations.
- Better coverage than binary-only implementations.

Conclusions

Trustworthy, Effective WCET Analysis.

- Built on trustworthy formal apparatus.
- Complete coverage of seL4 (with manual intervention).

http://ssrg.nicta.com.au/software/TS/wcet-tools

Future work:

- Verify more real-time adjustments to seL4.
- Improve TV/WCET interface.
- Support a more modern processor.